**WhisperXSegmentMerger.py – Usage and Design Notes**

**Bigya Bajracharya; ed. Phil Pfeiffer**

**Last Updated: May 18, 2024**

Table of Contents

[1. Introduction 1](#_Toc163696789)

[1.1 Purpose 1](#_Toc163696790)

[1.2 System Requirements 1](#_Toc163696791)

[2. Operation 1](#_Toc163696792)

[2.1 Overview 1](#_Toc163696793)

[2.2 Inputs 2](#_Toc163696794)

[2.3 Outputs 2](#_Toc163696795)

[2.4 Command Line Arguments 2](#_Toc163696796)

[3. Internal Design: Notes 2](#_Toc163696797)

[3.1 Architecture 2](#_Toc163696798)

# Introduction

## Purpose

The WhisperXSegmentMerger app merges adjacent transcribed segments from an individual speaker in a WhisperX output into individual segments.

Various options can be used to parameterize WhisperXSegmentMerger’s operation, including the names of its input and output files, whether it generates logs, and the name of its log files.

## System Requirements

* **Python Environment**: CoreAudioProcessor requires Python 3.12 or later to run. Python can be obtained for free from the official Python website, <https://www.python>.org
* **Required PyPi Packages**
  + **xmlschema**: For using schemas to validate XML content.
* **Other**
  + **CoreAudioProcessor**: The supporting file that transcribes and diarizes recordings.

# Operation

## Overview

As input, WhisperXSegmentMerger accepts the name of a file that was generated by the CoreAudioProcessor and optional, assorted parameters for configuring its operation. WhisperXSegmentMerger then condenses the output from CoreAudioProcessor, collapsing sequences of utterances from a single speaker into individual, extended sequences.

As part of this process, WhisperXSegmentMerger, by default, produces a log that records formatting anomalies in its input. Most commonly for the WhisperXSegmentMerger, these will be “?” for speaker names.

WhisperXSegmentMerger offers four ways to customize its operation:

* **Baseline Configuration**: This is a set of default settings built into the program.
* **Configuration File**:
  1. CoreAudioProcessor provides an XML file that can be changed to override the default settings in the baseline configuration.
  2. Alternatively, a configuration file can be specified on an execution’s command line. These configuration-file’s values will override the values specified in the CoreAudioProcessor’s built-in configuration file and its baseline configuration.
* **Command-line Parameters**: CoreAudioProcessor also accepts options on its command line. One such option, the source file, is required. Other options, if specified, override CoreAudioProcessor’s default and configuration file options.

## Inputs

* **Transcript**: The file to transcribe.
* **Command Line Parameters:** These parameters, which control how WhisperXSegmentMerger operates, are described below. More information can be obtained from WhisperXSegmentMerger’s header comment.

## Outputs

WhisperXSegmentMerger, if successful, creates a transcription of the specified audio recording in the specified output directory. If enabled, it also diarizes this recording and logs application activity and errors. Final performance data is logged if enabled.

## Command Line Arguments

WhisperXSegmentMerger accepts the following command-line arguments:

* + **-tr, --transcript**: Path to the CoreAudioProcessor file to reformate. (Required)
  + **-cx, --config\_processor**: Path to an alternative XML configuration file.
  + **-el, --enable\_logfile**: If true, enable logfile output.
  + -**ld, --logfile\_dir**: Directory to which to write the log file.
  + **-ln, --logfile\_name**: Name of the log file.
  + **-od, --output\_dir**: Directory for storing the transcriptions.
  + -**of, --ouput\_file**: Name of the output file.

# Internal Design: Notes

## Architecture

WhisperXSegmentMerger.py is structured as a stack of the following modules, which have been layered as shown below:

* main.py: WhisperXSegmentMerger’s entry point. Its header comment describes the application’s parameters, giving design notes.
* SegmentMerger.py: Handles the segment merging process.
* MergerConfig.py: Handles the configuration settings for the application. It reads and validates configuration parameters from various sources, such as command-line arguments, configuration files, or default settings, and ensures that transcription-related parameters are properly configured.
* CommandLineParser.py: Provides command-line parsing for the application using the argparser library.
* XMLProcessor.py: Deals with reading and writing XML files, particularly for configuration purposes. It provides functionalities to parse and manipulate XML data, facilitating the management of configuration settings.
* LogWrapper.py: The LogWrapper class is used to configure and initialize logging with predefined settings. It provides logging functionality to monitor and troubleshoot the application. It logs various events, errors, and performance data, aiding in debugging and understanding the application's behavior.
* DEFAULTS.py: Contains definitions for values and settings for an audio transcription system, including paths, file extensions, logging configurations, model types, and other constants for the system's default behavior.
* CONSTANTS.py: Contains values for various keys used throughout the application. These constants include values for default settings, logging codes, or other keys that are reused across modules.
* StatusManager.py: Manages the overall status and errors of the application. It supports the generation of program status messages.